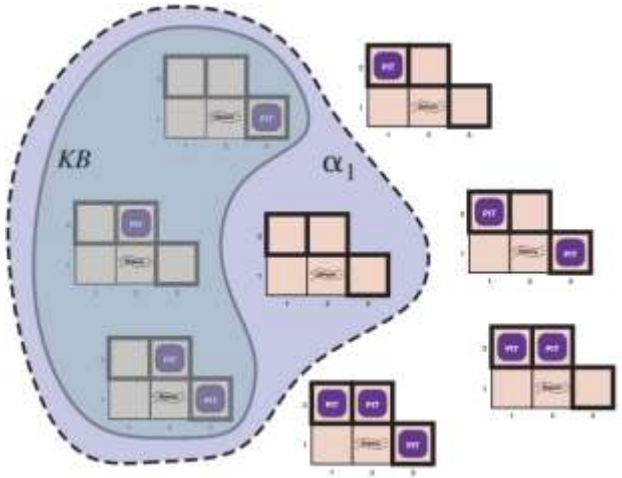
Inference Algorithms

ITI0210, lecture 7 (2021)

Review

KB – current knowledge, many possible worlds
(we don't know everything)



α – some sentence, true in all of them
(we know **enough** to be **sure** that α can be deduced from KB)

Generally: $KB \models \alpha$

Propositional logic: $KB \wedge \neg\alpha$ is unsatisfiable (contradiction)

Theorem Proving

Deriving New Sentences using Rules

Proof Systems

Example from Wikipedia deducing that if $A \wedge B$, then $B \wedge A$.

$$\dfrac{\dfrac{A \wedge B \text{ true}}{B \text{ true}}\ \wedge_{E2} \qquad \dfrac{A \wedge B \text{ true}}{A \text{ true}}\ \wedge_{E1}}{B \wedge A \text{ true}}\ \wedge_{I}$$

Uses the "Λ elimination" (E1,E2), "Λ introduction" (I) rules

General idea: can make new sentences using **rules of inference**

Inference vs Model Checking

Do we really need to check $2^8 = 256$ combinations of truth values?

It is obvious that H **must** be true

in all worlds where the KB is true

Do a **symbolic** proof, e.g.

from A and $A \rightarrow B$ we can get B

(not currently in KB!), etc until H is derived

Example knowledge:

$$A$$
$$A \rightarrow \mathrm{B}$$
$$B \rightarrow C$$
$$C \rightarrow D$$
$$D \rightarrow E$$
$$E \rightarrow F$$
$$F \rightarrow G$$
$$G \rightarrow H$$

Want to prove H

Inference Algorithms

We will cover:

- Forward chaining, using *modus ponens*
- Resolution

Forward Chaining

Fast algorithm for simpler KB-s

Forward Chaining

We have a KB. Want to know if some symbol Q (query) is true.

Apply *modus ponens* repeatedly to the contents of KB

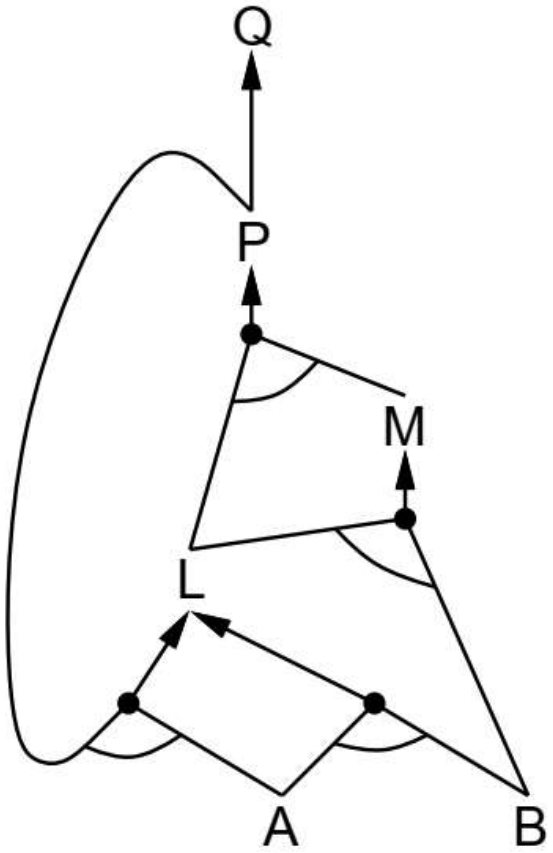
$$\frac{\alpha \rightarrow \beta \quad \alpha}{\beta}$$

See if Q is eventually derived

Forward Chaining

Example knowledge

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Forward Chaining

Implementation: a table to keep track of what can be derived next

Clause	Symbol	Premises	Count	Inferred
$P \rightarrow Q$	Q	P	1	F
$L \wedge M \rightarrow P$	P	L, M	2	F
$B \wedge L \rightarrow M$	M	B, L	2	F
$A \wedge P \rightarrow L$	L	A, P	2	F
$A \wedge B \rightarrow L$	L	A, B	2	F
A	A		0	F
B	B		0	F

Forward Chaining

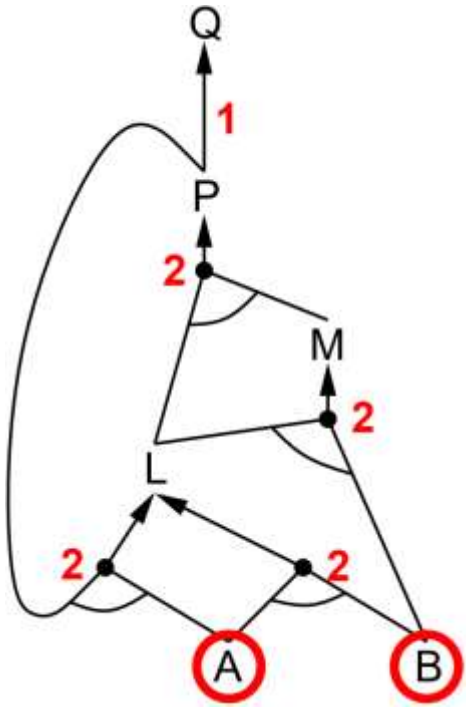
Implementation: a table to keep track of what can be derived next

Clause	Symbol	Premises	Count	Inferred
$P \rightarrow Q$	Q	P	1	
$L \wedge M \rightarrow P$	P	L, M	2	F
$B \wedge L \rightarrow M$	M	B, L	2	F
$A \wedge P \rightarrow L$	L	A, P	1	F
$A \wedge B \rightarrow L$	L	A, B	1	F
A	A		0	T
B	B		0	F

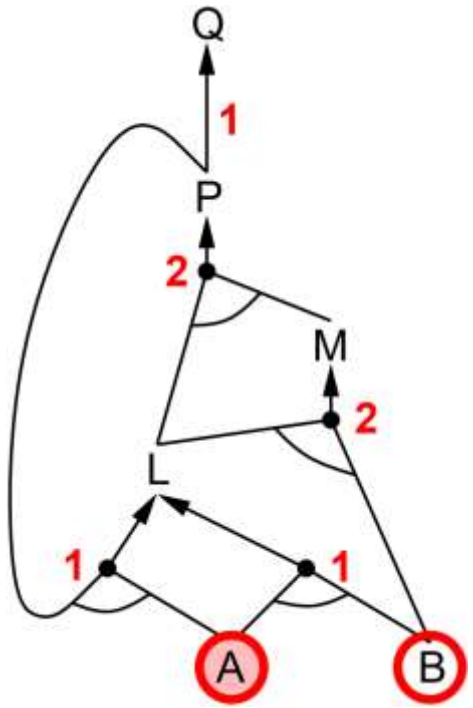
Remove premise, decrement count

0 premises to prove, is inferred

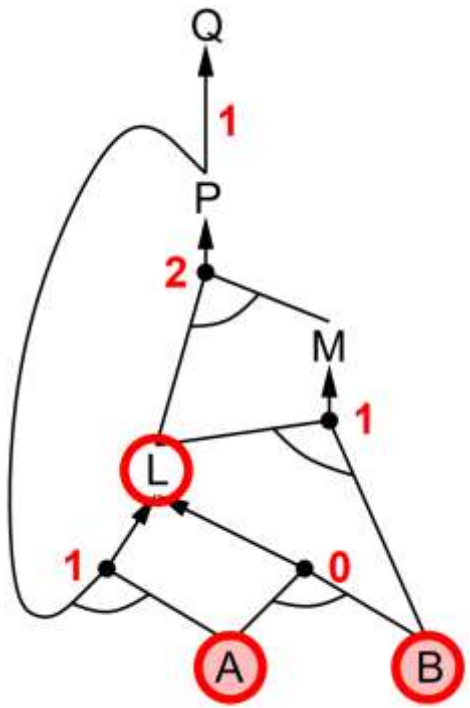
Forward Chaining



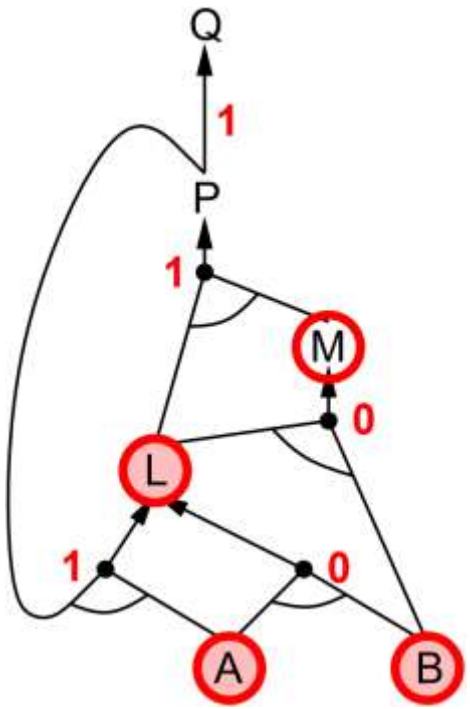
Start



A inferred



B inferred



L inferred
M next, etc

Forward Chaining

"Clause" – KB consists of $clause_1 \wedge clause_2 \wedge \cdots \wedge clause_n$

Each is a **Horn clause** $A \wedge B \wedge \cdots \wedge N \rightarrow Q$

equivalent to $\neg A \vee \neg B \vee \cdots \vee \neg N \vee Q$

Exercise: how is it equivalent?

Forward Chaining

Pros: fast, even simplest implementation will run $O(nm)$

n – number of symbols

m – KB size

With indexing, runs linear $O(m)$

Cons: Horn clauses cannot express everything

e.g. you cannot add $Rain \vee Fog \vee Snow$ to KB

Resolution

General inference procedure

Resolution Rule

Rewrite *modus ponens*:

$$\frac{\alpha \rightarrow \beta \quad \alpha}{\beta} \qquad \frac{\neg\alpha \vee \beta \quad \alpha}{\beta}$$

Negative and positive literal

Resolution is a more general rule, does the same thing:

$$\frac{l_1 \vee \cdots \vee l_i \vee \cdots \vee l_k \qquad m_1 \vee \cdots \vee m_j \vee \cdots \vee m_n}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

This mega rule just deletes l_i and m_j and joins everything else

l_i and m_j must be matching positive and negative literal, e.g. α and $\neg\alpha$

Resolution Rule

$$\frac{l_1 \vee \cdots \vee l_i \vee \cdots \vee l_k \qquad m_1 \vee \cdots \vee m_j \vee \cdots \vee m_n}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

l_i deleted from here

join clauses

m_j deleted from here

l –s and m –s are just symbols, can be either sign

l_i and m_j must be matching positive and negative literal, e.g. α and $\neg\alpha$

Resolution Rule

Example clauses:

$$A \vee B \vee \neg C \vee E$$
$$\neg B \vee C \vee D \vee E$$

Two matching pairs, **pick one**

$$\neg C \text{ and } C$$

We get the **resolvent**:

$$A \vee B \vee \neg B \vee D \vee E$$

E was present in both clauses, don't have to add it twice

Tautology

Side note: this resolvent is actually always true

$$A \vee B \vee \neg B \vee D \vee E$$

This happens in propositional logic

when we have **more than one pair** of opposite sign literals

(We will use this simplification later)

CNF

Resolution requires that the KB is in Conjunctive Normal Form (CNF)

$$\alpha_1 \wedge \alpha_2 \wedge \cdots \wedge \alpha_n$$

Where each α_i is a logical OR of symbols, e.g. $A \vee B \vee \neg P$

Example of CNF:

$$(\neg A \vee B) \wedge (A \vee C) \wedge (D \vee \neg C \vee \neg B)$$

So KB is a:

1. one big logical formula
2. can also view as **set of clauses**

Contradictions

Consider this formula

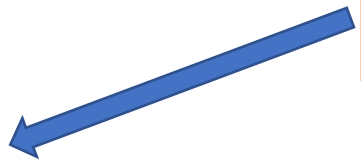
$$(\neg A \vee B) \wedge D \wedge (D \vee \neg C \vee \neg B) \wedge \neg D$$

Contains contradiction

$$D \wedge \neg D$$

Resolution can find this

$$\frac{\neg\alpha \quad \alpha}{\square}$$



nothing remains,
empty clause
= contradiction

Resolution method

Idea level algorithm:

try to generate new clauses
by combining pairs of clauses

Possible outcomes:

- contradiction in $KB \wedge \neg\alpha$
- or, cannot generate any more clauses

```
def resolution(KB, alpha)
    clauses = KB.union([neg(alpha)])
    while True:
        new = set()
        for cl1 in clauses:
            for cl2 in clauses:
                # all possible variant
                resolvents = resolve_rule(cl1, cl2)
                for r in resolvents:
                    if not r: # empty clause
                        return True  # contradiction
                new.update(resolvents)
        if new.issubset(clauses):
            return False # nothing new generated
        clauses.update(new)
```

Resolution method

Idea level algorithm:

try to generate new clauses
by combinin

Possible outc

- contradiction in $KB \wedge \neg\alpha$
- or, cannot generate any more clauses

Warning: this is how resolution works in principle
the listed function is super inefficient
better version (ANL loop) given with homework

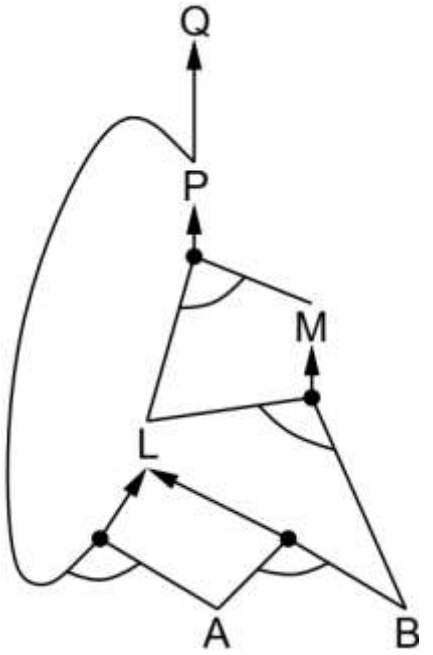
```
def resolution(KB, alpha)
    clauses = KB.union([neg(alpha)])
    while True:
        new = set()
                                        riant
                                    lve_rule(cl1, cl2)
                                    ts:
                                    mpty clause
                        return True  # contradiction
                new.update(resolvents)
        if new.issubset(clauses):
            return False # nothing new generated
        clauses.update(new)
```

Examples

Example 1: Horn clauses

Input clauses: $KB \wedge \neg \boldsymbol{\alpha}$

$$(\neg P \vee Q) \wedge (\neg L \vee \neg M \vee P) \wedge (\neg B \vee \neg L \vee M) \wedge$$
$$(\neg A \vee \neg P \vee L) \wedge (\neg A \vee \neg B \vee L) \wedge A \wedge B \wedge \neg \boldsymbol{Q}$$



$A \wedge B \rightarrow L$
converted

Query clause added

Examples

Inputs:

$\neg P \vee Q$

$\neg L \vee \neg M \vee P$

$\neg B \vee \neg L \vee M$

$\neg A \vee \neg P \vee L$

$\neg A \vee \neg B \vee L$

A

B

$\neg Q$

Step 1:

$\neg B \vee L$

$\neg P \vee L$

$\neg A \vee \neg B \vee \neg M \vee P$

$\neg L \vee \neg M \vee Q$

$\neg A \vee L$

$\neg L \vee M$

$\neg A \vee \neg B \vee M$

$\neg P$

$\neg A \vee \neg P \vee \neg B \vee M$

$\neg B \vee \neg L \vee P$

Step 2 generates, among others: L and $\neg L \vee P$

Step 3 generates $\neg L$, leading to contradiction (meaning, proof found)

Examples

Forward chaining proof path

Inputs:

$\neg P \vee Q$

$\neg L \vee \neg M \vee P$

$\neg B \vee \neg L \vee M$

$\neg A \vee \neg P \vee L$

$\neg A \vee \neg B \vee L$

A

B

$\neg Q$

Step 1:

$\neg B \vee L$

$\neg P \vee L$

$\neg A \vee \neg B \vee \neg M \vee P$

$\neg L \vee \neg M \vee Q$

$\neg A \vee L$

$\neg L \vee M$

$\neg A \vee \neg B \vee M$

$\neg P$

$\neg A \vee \neg P \vee \neg B \vee M$

$\neg B \vee \neg L \vee P$

Step 2 generates, among others: L and $\neg L \vee P$

Step 3 generates $\neg L$, leading to contradiction (meaning, proof found)

Examples

Inputs:

$\neg P \vee Q$ $\quad \neg A \vee \neg P \vee L$ $\quad A$

$\neg L \vee \neg M \vee P$ $\quad \neg A \vee \neg B \vee L$ $\quad B$

$\neg B \vee \neg L \vee M$ $\quad \neg Q$

Backward chaining path to $\neg L$

Step 1:

$\neg B \vee L$ $\quad \neg A \vee L$ $\quad \neg P$

$\neg P \vee L$ $\quad \neg L \vee M$ $\quad \neg A \vee \neg P \vee \neg B \vee M$

$\neg A \vee \neg B \vee \neg M \vee P$ $\quad \neg A \vee \neg B \vee M$ $\quad \neg B \vee \neg L \vee P$

$\neg L \vee \neg M \vee Q$

Step 2 generates, among others: L and $\neg L \vee P$

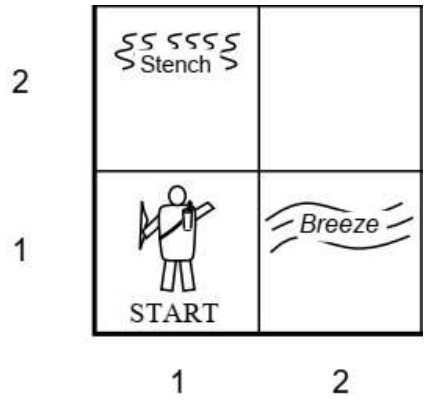
Step 3 generates $\neg L$, leading to contradiction (meaning, proof found)

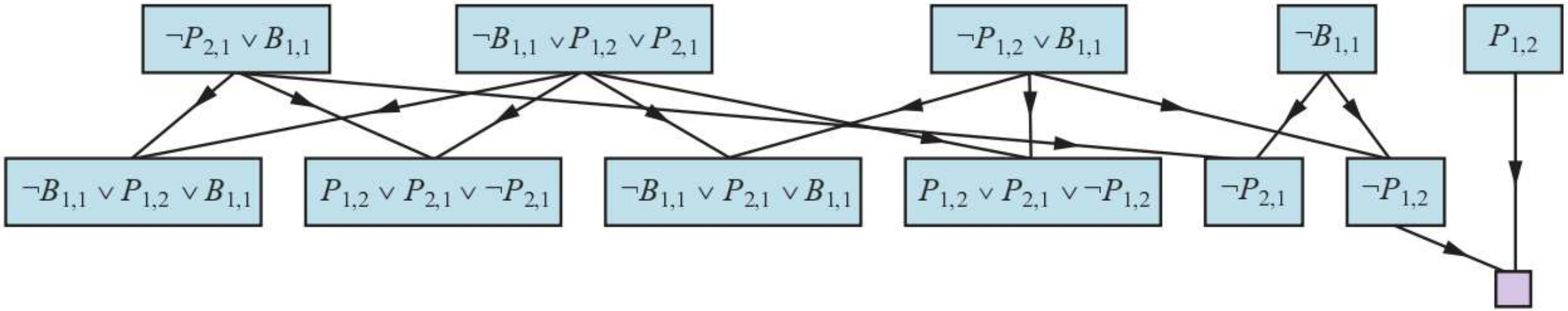
Examples

Example 2: Wumpus World

Partial KB: $\left(B_{1,1} \Leftrightarrow \left(P_{1,2} \vee P_{2,1}\right)\right) \wedge \neg B_{1,1}$

α is $\neg P_{1,2}$





CNF conversion

Resolution requires CNF. Conversion in propositional logic:

1. Eliminate $\Leftrightarrow$ by replacing $\alpha \Leftrightarrow \beta$ with $\alpha \rightarrow \beta \wedge \beta \rightarrow \alpha$
2. Eliminate $\rightarrow$ by replacing $\alpha \rightarrow \beta$ with $\neg\alpha \vee \beta$
3. Move $\neg$ inwards using de Morgan and double negation
4. Apply distributivity law $(\alpha \wedge \beta) \vee \gamma \equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma)$
5. Flatten $\big(\alpha \wedge (\beta \wedge \gamma)\big) \equiv \alpha \wedge \beta \wedge \gamma$

CNF conversion

Getting the CNF clauses in resolution Example 2:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$B_{1,1} \rightarrow (P_{1,2} \vee P_{2,1}) \wedge (P_{1,2} \vee P_{2,1}) \rightarrow B_{1,1}$$
$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(\boldsymbol{P_{1,2} \vee P_{2,1}}) \vee B_{1,1})$$
$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg \boldsymbol{P_{1,2}} \wedge \neg \boldsymbol{P_{2,1}}) \vee \boldsymbol{B_{1,1}})$$

result:

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$$

CNF conversion

Or… go to http://logictools.org/prop.html